intel

APPLICATION NOTE

AP-183

August 1984

8256AH Multifunction Peripheral Simplifies Microcomputer I/O Design

CHRISTOPHER SCOTT

Order Number: 231125-001

8256AH Multifunction Peripheral Simplifies Microcomputer I/O Design

CONTENTS

231125-001

INTRODUCTION

A primary goal of microcomputer system design is to provide the required functionality and flexibility with the fewest number of components. The 8256AH Multi-function Peripheral is designed specifically to meet these conflicting requirements. Four of the most common microcomputer system functions, previously requiring up to four separate MSI or LSI devices, are combined into one LSI device. The 8256AH incorporates a serial asynchronous communication channel, two 8-bit parallel I/O ports, five 8-bit timer/counters and an eight level priority interrupt controller in one 40 pin package. Its flexible design allows it to directly interface to most microprocessors, including Intel's MCS-85, iAPX-86, iAPX-88, iAPX-186 and iAPX-188, and the MCS-48 and MCS-51 family of single-chip microcomputers.

This application note describes using the 8256AH to implement a Data Terminal Equipment (DTE) RS-232C serial asynchronous communication link with the control signals necessary to interface to a Bell 103/212A modem. The interface requires a total of nine interface signals. Three of these signals, TxD, RxD and CTS, are provided by the UART section of the 8256AH. The balance of the RS-232C interface signals are implemented using six of the independently programmable parallel PORT 1 lines. In addition, the application design provides an eight bit parallel I/O port with handshaking signals. The on-chip priority interrupt controller enables the RS-232C serial interface an the parallel interface to operate on an interrupt basi The 8256AH uniquely addresses the complexities of implementing an RS-232C communications interface. By utilizing the built-in hardware and software features of the 8256AH, the design achieves flexibility with simplicity, qualities often exclusive of one another.

Previous solutions required four components to implement the same interface. Figure 1 illustrates the basic system block diagrams for the two solutions. In Figure 1a the 8251A Programmable Communications Interface provides the UART serial communications interface. The 8254 Programmable Interval Timer provides baud rate generation and other timing functions, such as time-out loops, needed for software support of an RS-232C interface. These are especially needed if the RS-232C channel is to operate in an interrupt system environment. The 8255A Programmable Peripheral Interface provides parallel I/O with one port dedicated to the RS-232C control signals. The 8259A Priority Interrupt Controller provides an eight level priority interrupt structure. This represents a total of 120 device pins compared to the single 40 pin 8256AH, and 465 mA current requirement verses a 160 mA current requirement. Figure 1b represents the 8256AH solution incorporating the four functions in one package.

In some data communication applications only three lines - ground, Transmit Data and Receive Data - are used for serial communication. An example is communication between an ASCII terminal or printer and a personal computer. These devices are usually located close to one another and in general do not require the additional control signals of the EIA RS-232C serial communications standard. In other data communications applications, this same equipment requires that the integrity of the serial communications link be constantly monitored. This enables the host system to control the data trransmission at all times, whether it be a host computer or intelligence local to a communications device, such as an ASCII terminal. The need for control and monitoring of the serial line is particularly important when the communications link is over telephone lines using a modem. In a Switched Network, where a number of serial devices share the same communications line, the control signals are crucial to the system's multiplexing the single line.

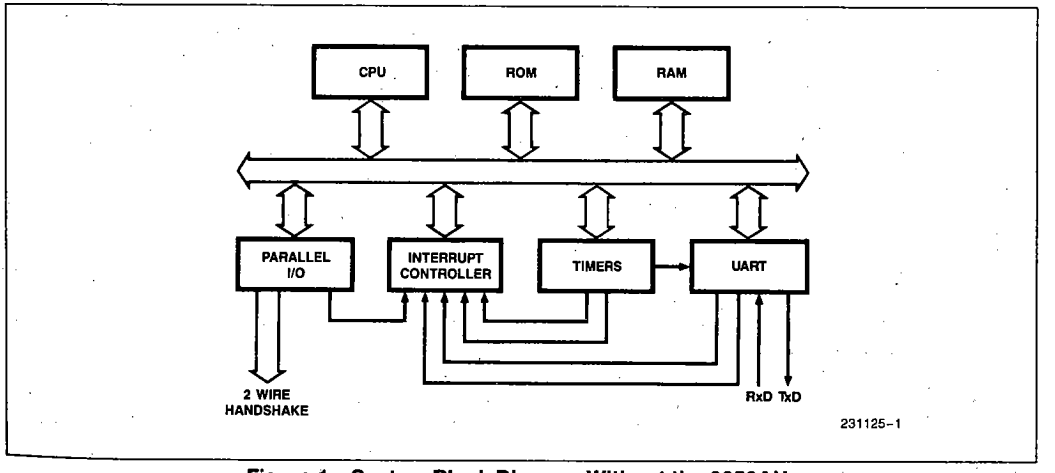

Figure 1a. System Block Diagram Without the 8256AH

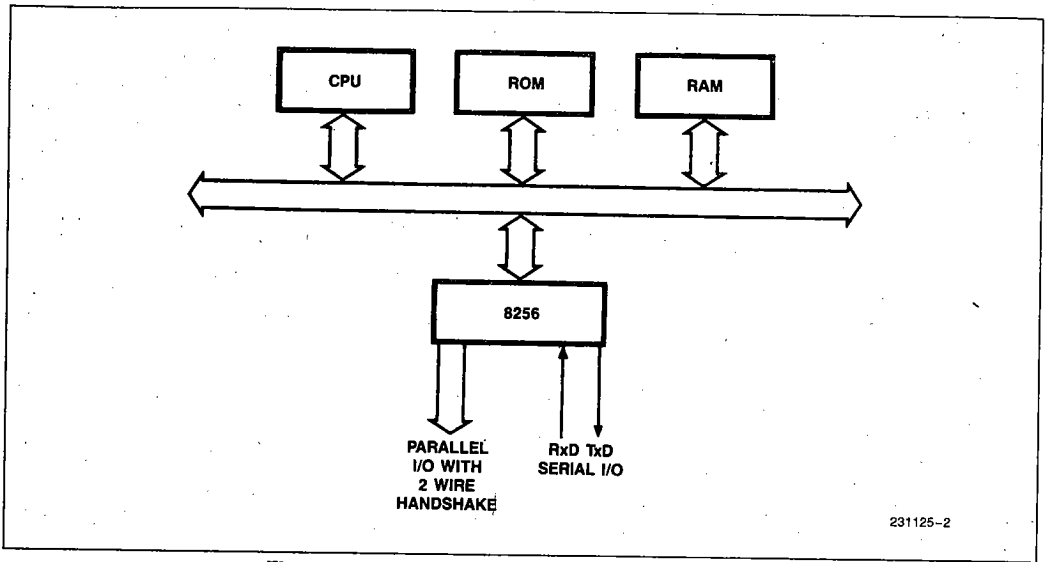

Figure 1b. System Block Diagram With the 8256AH

This Application Note assumes that the reader is familiar with the 8256AH Data Sheet and with the RS-232C communication protocol and terminology. A complete software listing is provided in Appendix A. A complete description and definition of the RS-232C interface standard may be found in the book "Data Communications: A Users Guide" by Kenneth Sherman, Reston Publishing 1981.

DESCRIPTION OF THE 8256AH

The 8256AH combines four commonly used peripheral functions into one device (see Figure 2);

1. A full-duplex, double-buffered serial asynchronous Receiver/Transmitter (UART) with an on-chip Baud Rate Generator.

2. Two 8-bit parallel I/O ports; One bit programmable, One nibble programmable.

3. Five 8-bit timer/counters; 4 can be cascaded to form 2 16-bit timer/counters

4. An 8-level priority interrupt controller.

The 8256AH uses the standard bus control signals compatible with Intel's family of peripherals and microprocessors. The microprocessor interface utilizes a multiplexed address/data bus. Four of the eight address/data lines are used to generate the register address. This enables all of the 8256AH's functionality to be contained in a 40 pin package while retaining direct register addressing.

The sixteen directly addressable internal read/write registers provide control for all of the 8256AH's various functions. Fourteen of the registers are read/write, one, the Status Register, is read only and one, the Modification Register, is write only. Three Command Registers configure the operating environment including the type of CPU, 8 or 16 bit, and system clock frequency. Command Register Three provides bit set-reset capability for control of such functions as End of Interrupt, Nested Interrupts, Interrupt Acknowledge and UART Receive Enable. The Status Register provides all information about the UART's transmitter and receiver, and the state of the interrupt (INT) output pin to the microprocessor. The Mode Register defines the configuration of the two parallel ports and the five timer/counters. The write only Modification Register is used to alter two standard functions of the receiver, start bit sampling and to enable a special indicator flag for half-duplex operation. In addition, six registers control the two parallel ports. Two registers provide for UART Transmit and Receive Buffers. Ten registers are used for timer/counter interface, and four registers provide for Priority Interrupt Controller support.

The UART section of the 8256AH features a full-duplex double-buffered transmitter and receiver with separate control registers. The internal baud rate generator provides the thirteen common sampling rates from 50 bps to 19.2 kbps. An external baud rate clock can also be used, with programmable choice of 1X, 32X or 64X sampling rates.

The two parallel I/O ports can be configured as two independent 8-bit parallel I/O ports, or as one 8-bit

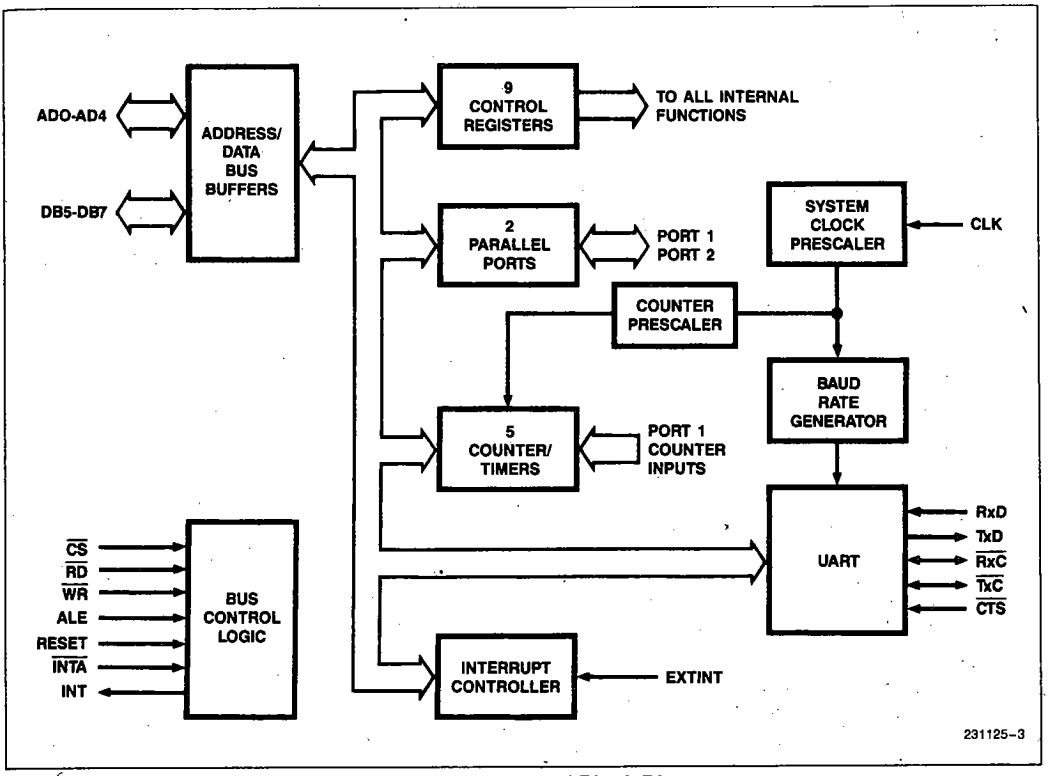

Figure 2. 8256AH Internal Block Diagram

parallel port with ACK/OBF and STB/IBF two wire handshake signals. In the latter configuration, the six remaining I/O lines may be used as either independently programmable I/O lines, or as predefined special function inputs and/or outputs, such as a second external interrupt input or timer/counter inputs.

The five 8-bit programmable timer/counters are binary presettable downcounters. In addition, an independent on-chip Baud Rate Generator is provided for the UART. The clock sources for the timers/counters may be either internal or external - via programmed parallel port pins - depending upon whether they are configured as timers or counters. Four of the timer/counters may be cascaded to form two 16-bit timer/counters. Each of the five timer/counters has its own read/write register.

The eight level priority interrupt controller has twelve possible interrupt sources. Ten of the sources are internal and two are external. One of the external interrupt sources is a fixed pin; EXTINT. The second is one of the parallel Port 1 pins which can be programmed as an external interrupt source. The twelve interrupt sources are internally mapped to the eight interrupt priority levels.

The interrupt controller may be programmed to operate in either a Normal or Nested Interrupt Mode. In Normal Mode any interrupt may interrupt any other interrupt based upon the enable/disable bits in the Interrupt Enable, or Mask, Register. In the Nested Mode only an interrupt of higher piority may interrupt one of lower priority, again based upon the bits in the Enable Register.

The 8256AH interrupt structure supports both 8085 and 8086 interrupt vectoring methods via the INTR and INTA signals. In vectored interrupt operation the 8256AH places the interrupt vector address on the data bus during the INTA sequence. In addition the 8256AH supports non-vectored interrupt interfaces, such as MCS-51 and MCS-48 systems. In non-vectored interrupt applications the host system simply reads the interrupt vector address from the Interrupt Address Register of the 8256AH. Reading the interrupt address register clears the INT pin and acknowledges that the interrupt has been serviced. This is the functional equivalent to an INTA sequence generated by the host processor.

DESIGN DESCRIPTION

Hardware Description

Figure 3 shows a block diagram of this application's system design. The microprocessor used is an iAPX-186 with two 8256AH's for parallel and serial I/O, as well as for providing a variety of system support functions. One 8256AH is used to implement both the RS-232C modem interface and provide multiplexed parallel I/O. The system uses the Intel 957B System Monitor for control of the system hardware and software development support. The second 8256AH is used for basic serial communication between an ASCII terminal and the Intel 957B System Monitor residing in 16K bytes of EPROM. The two 8256AHs provide a total of six I/O channels - two UARTs and four parallel I/O ports.

When one of the 8256AHs is configured for the serial RS-232C interface, one of its parallel ports, Port 1 pins 2-7, provides control signals for the serial interface. Four of the RS-232C control signals (CTS, DSRS, DSR and CD) are OR'd to the EXTINT pin of the 8256AH. If any of these signals change from their defined state, an interrupt is generated to the 8256AH. The modem driver software then responds to the interrupt by reading the Port 1 register, determines the signal generating the interrupt and responds accordingly (see the software listing; INT—MOD). In addition to the RS-232C control signals, the communications software can support all of the standard UART error conditions such as framing errors, underrun, overrun and parity, if parity is enabled.

Parallel I/O With Handshaking

The remaining two Port 1 lines, not used for the RS-232C control signals, provide ACK/OBF and STB/IBF handshaking signals for parallel Port 2. In an environment which utilized the second parallel port, while implementing the above described RS-232C channel, both would operate on an interrupt basis. The interrupt software algorithm depends upon whether the parallel port is configured as input or output, and whether Nested or Normal interrupt mode is programmed. If Nested Interrupt Mode is used, the software flow would default to parallel input or output (as programmed) with Port 2 handshaking the lowest priority interrupt. The serial channel would then interrupt parallel Port 2 transmission whenever the serial channel transmitted or received a character. The RS-232C control signals, OR'd to the External Interrupt (EXTINT) pin, would have the highest interrupt controller priority. The Software Description below describes this in greater detail.

SOFTWARE DESCRIPTION

Serial RS—232C Interface

The software is written in PL/M and is broken up into four separate modules, each containing several procedures. A block diagram of the software structure is given in Figure 4. The modules are identified by the dotted boxes, and the procedures are identified by the solid boxes. Two or more procedures connected by a solid line means the procedure above calls the procedure below. The procedures without any solid lines connecting them are interrupt procedures. They are entered when the 8256AH interrupts the 80186 and vectors an indirect address to the 80186.

The Serial RS-232C Interface software uses nested interrupts. The priority of the interrupt procedures is given in Figure 5.

The priority of the interrupts is not programmable but they are logically oriented so that for this application the priority is correct. The serial receiver should have the highest priority since it could have overrun errors. Therefore the RxD request can interrupt any other interrupt service routine thus preventing any possibility of an overrun error.

The Serial RS-232C Interface software is entered via a GO instruction from the 957B System Monitor console. The software first calls POWR—ON—INIT which initializes the 8256AH. This sets the 8256AH to 8086 Mode with parallel Port 2 in two wire handshake mode using Port 1 pin 0-1 for Port 2 handshaking. The initialization configures six of the Port 1 lines, pins 2-7, for RS-232C handshaking—input or output depending upon the specific signal tied to the pin. Figure 6 illustrates the definition of each Port 1 RS-232C handshaking line and its direction.

Both the Serial RS-232C Interface and the parallel interface with handshaking operate on an interrupt basis. Following initialization the software enters an endless loop and awaits an interrupt from one of three sources; Receive Data (RxD), Transmit Data (TxD) or the parallel interface. In the serial interface idle state, neither transmitting nor receiving data, the software is constantly responding to TxD interrupts; a result of the Transmit Buffer (TBE) and/or Transmit Register (TRE) being continuely empty. When data is received by the RS-232C channel the RxD interrupt, being of higher priority, asserts its interrupt.

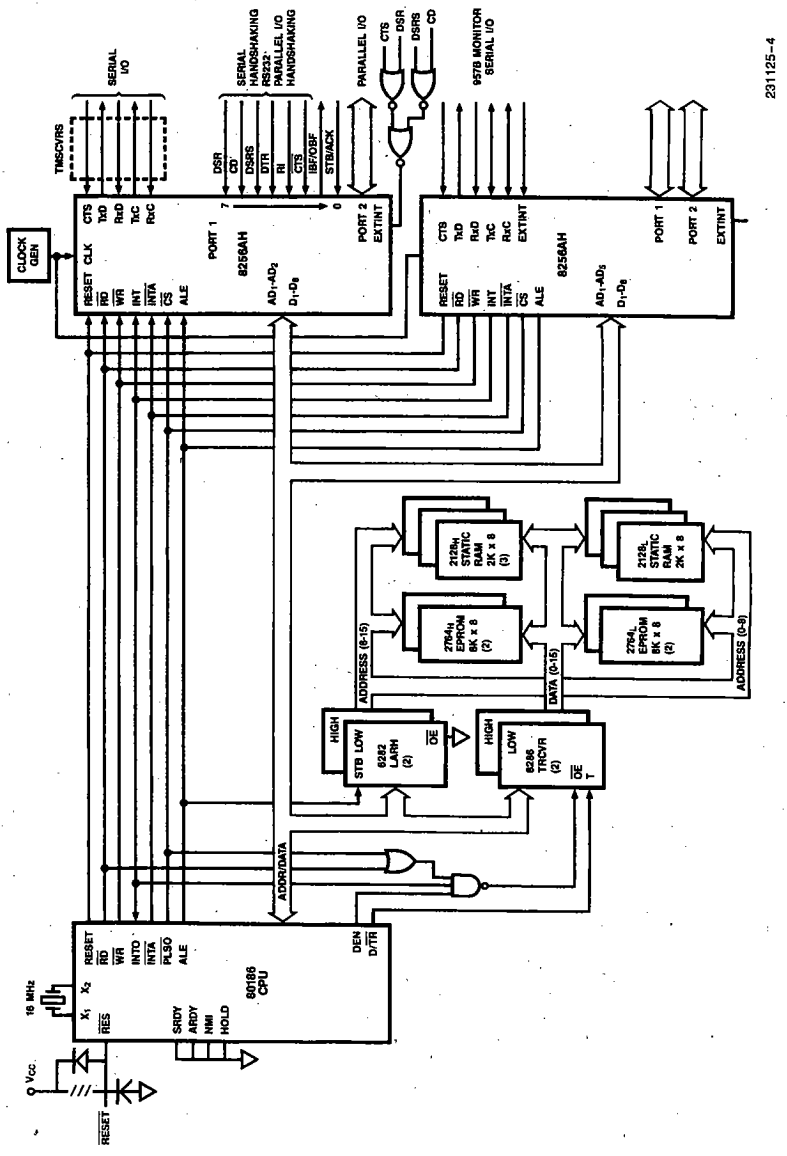

Figure 3. 8256AH / 80186 Schematic

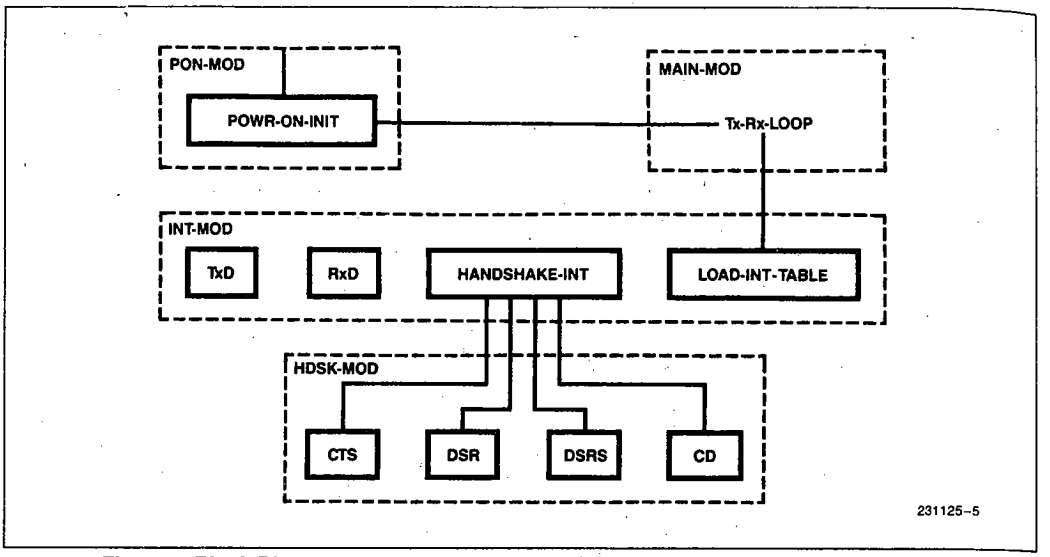

Figure 4. Block Diagram of the 8256AH Serial RS-232C Interface Software Structure

Priority		Source
Highest	0	Not Used
	1	Not Used
	2	External Interrupt (EXTINT)
	3	Not Used
	4	RxD Interrupt
	5	TxD Interrupt
	6	Timer 2 or 2 & 4 (16 bit)
	7	Port 2 Handshaking

Figure 5. 8256AH Interrupt Source To Priority Level Map

Port 1 Pin No.	Circuit	I/O	Abrev.	Signal Name
0			STB/ACK	Parallel Port 2
1			IBF/OBF	Handshaking Signals
2	CG	I	CTS	Clear To Send
3	CE	I	RI	Ring Indicator
4	CD	O	DTR	Data Terminal Ready
5	CI	I	DSRS	Data Signal Rate Selector
6	CF	I	RLSD (or CD)	Receive Line Signal Detector (Carrier Detect)
7	CC	I	DSR	Data Set Ready

Figure 6. Port 1 RS-232C Pin Definition

Although the parallel interface software is not implemented in the software listing of Appendix A, the algorithm for implementing multiplexed parallel and serial I/O is to input or output data on the parallel port during the relatively lengthy time required for serial communication overhead. The algorithm differs slightly during the serial channel idle state when the software responds to repetitive TxD interrupts. In this case the endless loop would detect the idle state repetitive TxD interrupts and disable the TxD interrupt for a short time while the parallel inputs or outputs data. This would require using one of the 8256AH timers to time out repetitive TxD interrupts. The timer used has to be lower in priority than the RxD interrupt to guarantee protection against overrun errors. Timer 2, or 2 and 4 cascaded if longer time delays are desired, provides the proper interrupt level as shown in Figure 5.

Figure 7 shows the Receive Data (RxD) interrupt service routine software flowchart. Since two conditions can generate an RxD Interrupt the Software first reads the Status Register and checks for the Break Detect (DB) bit being set. If the BD bit is clear, no Break condition being present, the data byte is read, stripped to seven bits, for an ASCII character, and sent to the system console via a call to the 957B System Monitor Console Output (CO) routine. Upon return from the 957B monitor call an End Of Interrupt (EOI) is sent to the 8256AH to reset the currently served interrupt level bit in the Interrupt Service Register.

Figure 8 shows the Transmit Data (TxD) interrupt service routine software flowchart. There are three conditions which may cause a TxD Interrupt; TBE, TRE and Break-In Detect. The TxD service routine first reads the Status Register to determine if the interrupt source is the TBE (Transmit Buffer Empty), if not then the interrupt service routine returns to the MAIN—MOD loop. If TBE = 1 (true) then a data byte is read from the 957B System Monitor Console Input (CI) routine. If the data byte is an ASCII character it is written to the 8256AH Transmit Buffer. The software exists via an EOI (End Of Interrupt) command to the 8256AH then returns to the MAIN—MOD Rx—Tx—Loop.

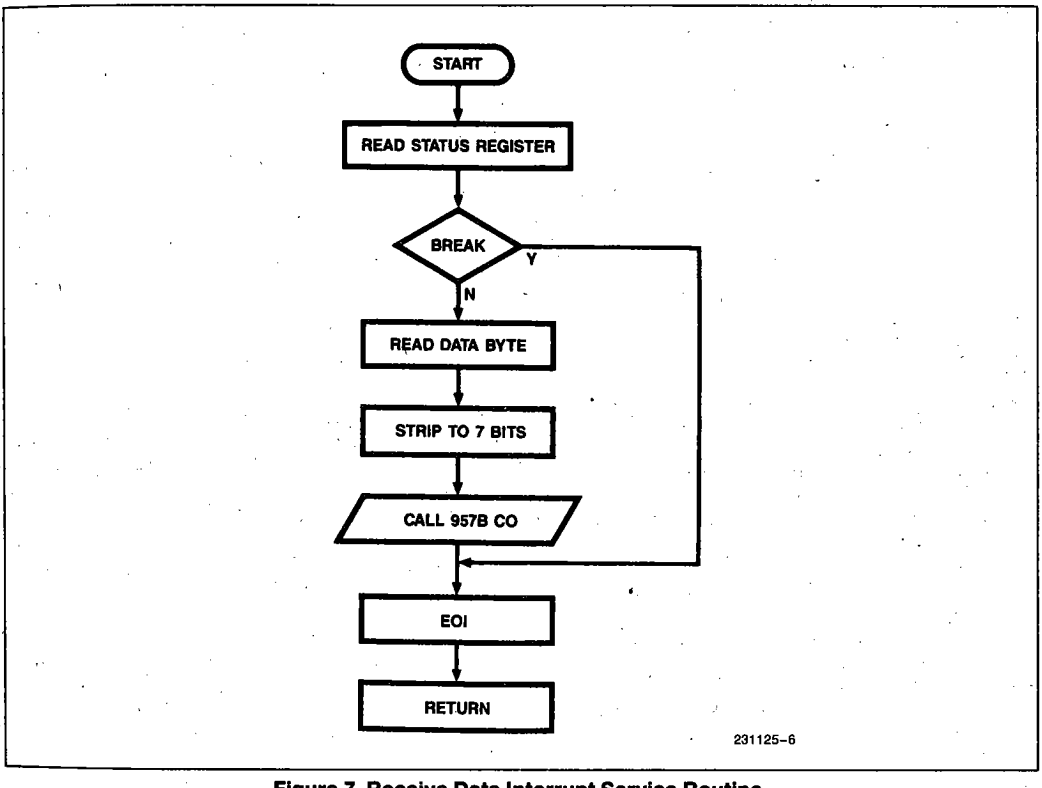

Figure 7. Receive Data Interrupt Service Routine Software Flowchart

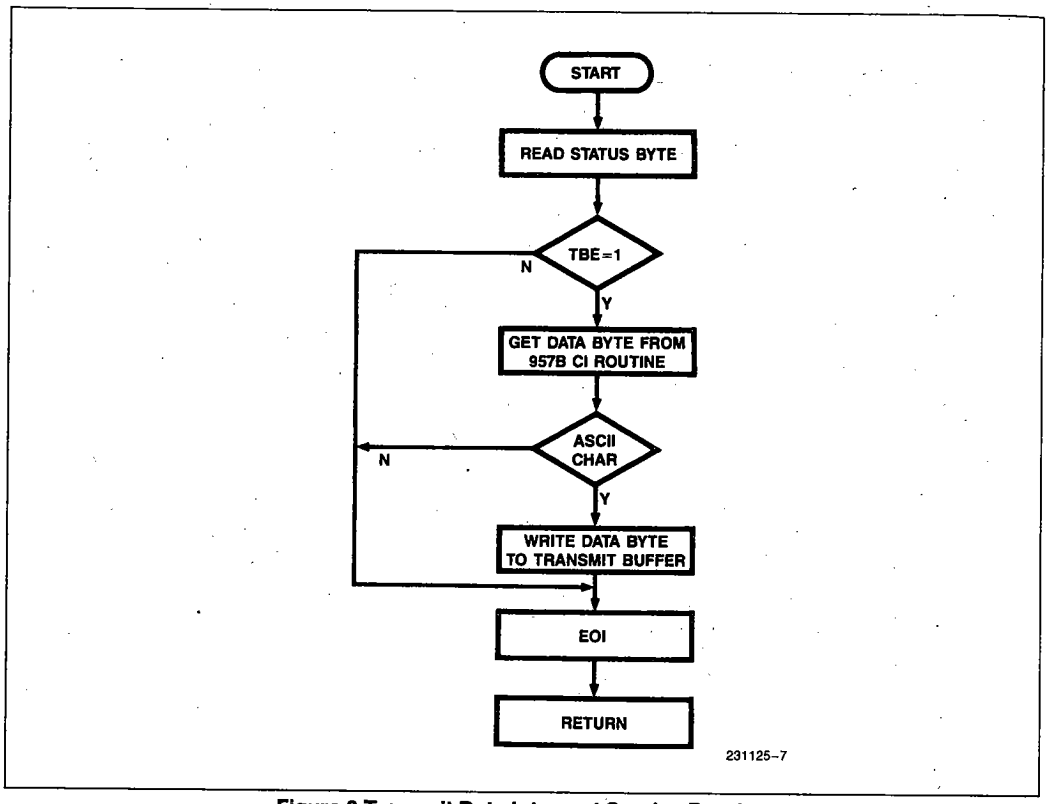

Figure 8.Transmit Data Interrupt Service Routine Software Flowchart

RS-232C Control Signals Interrupt Structure

The overall interrupt scheme is such that a change in a RS-232C handshake line causes an interrupt via the EXTINT pin on the 8256AH (see Figure 3 8256AH/80186 Schematic). The EXTINT interrupt is of higher priority than either the RxD or TxD interrupt. This enables the RS-232C handshake signals to manage the receipt or transmission of data via the nested interrupt mode of the 8256AH. The EXTINT interrupt service routine first reads the Port 1 pins 2-7 data and compares it to default state for the signal requiring service. The EXTINT interrupt service routine then calls the appropriate handshake signal service procedure as shown in the bottom module of Figure 4 Software Structure Block Diagram.

Each of the individual RS-232C control signal service procedures displays a message on the 957B monitor console device indicating the signal requiring a response. The service procedure then either initiates specific predefined actions or prompts the user with options. In a system which utilized file storage, such as a personal computer, the RS-232C software driver could pass a flag to the communications software, rather than a message. The communications software would in turn perform the same types of action but could also protect disk buffering files which might be open at the time of the interrupt. Two examples of the RS-232C Control Signal interrupt service routines, CTS and DSRS, are described below;

If Clear To Send (CTS) changes state, the UART automatically disables the transmitter. The CTS interrupt service procedure initializes the 8256AH's internal Timer. If the timer times out before CTS goes active again an interrupt is generated, a second message is displayed at the 957B monitor console prompting the user that the CTS line remains inactive. The options available at this point are to wait again, re-initializing Timer 1, or to disconnect the RS-232C channel.

If Data Signal Rate Selector (DSRS) changes state, the software prompts the user with a message that the Data Rates of the two RS-232C channels are not the same and the user is given the option of altering the data rate. This application example was interfaced to a 103A/212 Bell modem and as such prompts the user to select between 300 or 1200 bps data rates. In the case of a non-modem interface the routine could prompt the user for one of the thirteen standard data rates. The software then returns to the TxD/RxD software loop. The balance of the interrupt service procedures for the RS-232C handshaking signals function in a similar manner.

Depending upon the specific system design and software requirements, a variety of enhancements could be added to the system design. These could include interrupt traps that initiate specific corrective options or cascading multiple 8256AHs each with an RS-232C interfaces as described above. An example of an interrupt trap might be auto redial upon time out for lack of Carrier Detect (CD) upon initiating a communications link, or automatic disk file update when a receive buffer approaches overflow.

The ability of the 8256AH to be reprogrammed to meet the changing requirements of a system simplifies the overall system design and multiplies its capabilities. A simple reinitialization sequence could reconfigure the 8256AH as a UART with two parallel ports or utilize any of the various special functions of the parallel Port 1; e.g., an external timer input or an additional external interrupt input, etc. The reinitialization could also configure the 8256AH Multifunction Peripheral for a variety of custom applications.

CONCLUSION

The functional integration of the 8256AH makes it ideal for designs which require maximum flexibility and simplicity of implementation. The implementation of the RS-232C serial channel modem interface and multiplexed parallel I/O described in this application note represent a level of efficiency in peripheral performance and design previously unavailable. The 8256AH Multifunction Peripheral represents a savings of two-thirds the board space and power required by the previous four chip solution, with the added benefit of increased system reliability. The application note demonstrates the ease of implementing the variety of I/O capabilities and system support functions of the 8256AH. The integration of four common microprocessor system functions into one VLSI device enables the designer to devote valuable resources to adding features to enhance the system design, adding performance and flexibility, and reducing the system's overhead.

APPENDIX A.

SOFTWARE LISTING

```
PL/M-86 COMPILER     MAINMOD


SERIES-III PL/M-86 V2.3 COMPILATION OF MODULE MAINMOD
OBJECT MODULE PLACED IN :F2:56.OBJ
COMPILER INVOKED BY:  PLM86.86 :F2:56


                /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                *                                                                       *
                *         8256AH MULTIFUNCTION PERIPHERAL SIMPLIFIES                    *
                *            MICROCOMPUTER I/O DESIGN                                   *
                *                                                                       *
                *            Intel Corporation                                          *
                *            3065 Bowers Avenue                                         *
                *            Santa Clara,  Ca. 95051                                    *
                *                                                                       *
                *                                                                       *
                *                 Written By      Christopher Scott                     *
                *                                                                       *
                * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

                $MOD186 DEBUG LARGE
  1             MAINMOD:
                DO;

                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*     8256AH Register / Value / Constant  Definitions     */
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
  2   1         Declare Lit             Literally       'literally',
                      DCL               lit             'Declare',

                      True              lit             '0ffh',
                      False             lit             '0h',
                      Forever           lit             'while 1',
                      Pcs1              lit             '80h',
                      Cmd1reg           lit             'pcs1 + 0',
                      Cmd2reg           lit             'pcs1 + 2',
                      Cmd3reg           lit             'pcs1 + 4',
                      Modereg           lit             'pcs1 + 6',

                      Port1Ctrl1Reg     lit             'pcs1 + 8',
                      SetIntReg         lit             'pcs1 + 0ah',
                      EnIntReg          lit             'pcs1 + 0ah',
                      RstIntReg         lit             'pcs1 + 0ch',
                      IntAddrReg        lit             'pcs1 + 0ch',
                      TxBuffReg         lit             'pcs1 + 0eh',
                      RxBuffReg         lit             'pcs1 + 0ch',

                      Port1Reg          lit             'pcs1 + 10h',
                      Port2Reg          lit             'pcs1 + 12h',
                      Timer1Reg         lit             'pcs1 + 14h',
                      Timer2Reg         lit             'pcs1 + 1ah',
                      Timer3Reg         lit             'pcs1 + 1ch',
                      StatReg           lit             'pcs1 + 1eh',

                      Intr1             lit             'pcs1 + 40h',
                      Intr2             lit             'cs1 + 01h',
                      Intr3             lit             'pcs1 + 10h',
```

231125-8

231125-001

```
                Intr4           lit             'pcs1 + 08h',

                Int_Reset       lit             '88h',
                SioTxEn         lit             '10h',
                SioTxRdy        lit             '20h',
                SioRxRdy        lit             '40h',
                Break           lit             '04h',
                DisIntr         lit             '00h',
                StripTo7fh      lit             '7fh',
                Port1_Strip     lit             '0fcH',

                Cmd1            lit             '43h',
                Cmd2A           lit             '07h',
                Cmd2B           lit             '09h',
                Cmd3Clr         lit             '7fh',
                Cmd3            lit             '0a1h',
                Mode            lit             '00h',
                EnRcvr          lit             '0c0h',

                A               lit             '41h',
                B               lit             '42h',
                DSR             lit             '80h',
                DSR_Flag        lit             '80h',
                CD              lit             '40h',
                CD_Flag         lit             '40h',
                DSRS            lit             '20h',
                DSRS_Flag       lit             '20h',
                DTR             lit             '10h',
                RI              lit             '08h',
                CTS             lit             '04h',
                CTS_Flag        lit             '04h',

                (Status,
                 Hndshk_Pins,
                 J)             Byte,

                Char            Byte            External,

                Message_Ptr     Pointer;

        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
        /*          Message Declarations                                         */
3   1   DCL     CTS_MSG  (*) Byte Public Data ('CTS Disabled.  Receive Data stopped.',
                                               0AH,0DH,0),
            DSR_MSG  (*) Byte Public Data ('DSR Disabled.',0AH,0DH,00),
            CD_MSG   (*) Byte Public Data ('CD Disabled.',0AH,0DH,00),
            DSRS_MSG (*) Byte Public Data ('Enter Baud Rate:  A. 300   B. 1200
                                               (A/B)  : ',00),
            CTS2_MSG (*) Byte Public Data ('CTS Disabled.  Receive Data stopped.',
                                               0AH,0DH,00),
            Break_MSG (*) Byte Public Data ('Break in Receive Data.',0AH,0DH,00);
        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */



        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
        /*       External Procedures:                                           */
```

231125-9

```
                /*                                                              */
                /*          MCO:    957B Monitor Console Output Routine         */
                /*                                                              */
                /*          MCI:    957B Monitor Console Input Routine          */
                /*                                                              */
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

  4   1         MCO: Procedure(Char) External;
  5   2           DCL       Char        Byte;
  6   2         End MCO;

  7   1         MCI: Procedure Byte External;
  8   2         End MCI;

                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*      Initialize 8256AH Procedure                            */
  9   1         Init56: Procedure;

 10   2           Disable;
                  /*    Output 8256AH Init Data          */
 11   2           Output(Cmd1Reg)=Cmd1;          /* 8086 mode, freq=1khz, 1 stop bit,
                                                    and 7 bit char */
 12   2           Output(Cmd1Reg)=Cmd2A;          /* odd parity, system clk=1.024mhz,
                                                    and 1200 bps */
 13   2           Output(Cmd1Reg)=Cmd3C1r;       /* clear cmd reg 3 */
 14   2           Output(Cmd1Reg)=Cmd3;          /* reset, itr ack enabled, nested
                                                    intr mode  */
 15   2           Output(Cmd1Reg)=EnRcvr;        /* enable receiver */
 16   2           Output(Cmd1Reg)=Mode;          /* cascade timers 3&5, for the
                                                    reeiver$timer$out timer, byte &
                                                    output mode */
 17   2           Call Load_Int_Table;
 18   2           Enable;

 19   2         End Init56;
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*     Procedure:      Load Interrupt Address Vectors          */
 20   1         Load_Int_Table: Procedure Public;

 21   2           Call Set$Interrupt(42H,EXTINT);
 22   2           Call Set$Interrupt(44H,Receive_Char);
 23   2           Call Set$Interrupt(45H,Transmit_Char);
 24   2           Call Set$Interrupt(46H,Timer_2_4);

 25   2         End Load_Int_Table;
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*            EXTINT Interrpt Procedure:                       */
                /*                                                             */
```

231125-10

```
                /*              Service routine reads the Port 1 RS232          */
                /*              handshake signals and sets the message pointer  */
                /*              coresponding to the signal detected.            */
                /*                                                              */
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
 26   1         EXTINT: Procedure Interrupt 42H;

 27   2           Enable;
 28   2           HndShk_Pins=Input(Port1Reg) and Port1_Strip;
 29   2           If CTS_Flag = HndShk_Pins and CTS Then
 30   2             Do;
 31   3               Message_Ptr=@CTS_MSG(0);
 32   3               Output(Timer2Reg)=100;
 33   3             End;
 34   2           Else
                    If DSR_Flag = HndShk_Pins and DSR Then
 35   2               Message_Ptr=@DSR_MSG(0);
 36   2             Else
                      If CD_Flag = HndShk_Pins and CD Then
 37   2                 Message_Ptr=@CD_MSG(0);
 38   2               Else
                        If DSRS_Flag = HndShk_Pins and DSRS Then
 39   2                   DO;
 40   3                     Message_Ptr=@DSRS_MSG(0);
 41   3                     If MCI = A Then
 42   3                       Output(Cmd1Reg)=Cmd2A;        /* odd parity, system clk=1.024mhz,
                              Else                             and 1200 bps */
 43   3                       If MCI = B Then
 44   3                         Output(Cmd1Reg)=Cmd2B;      /* odd parity, system clk=1.024mhz,
                                                              and 300 bps */
 45   3                   End;
 46   2           Call Send_Msg;
 47   2           OutPut(RstIntReg)=Int_Reset;

 48   2         End EXTINT;
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*       Procedure       Receive a character                   */
 49   1         Receive_Char: Procedure Interrupt 44H;

 50   2           Enable;
 51   2           Status=(Input(StatReg) and SioRxRdy);
 52   2           If Status AND Break Then
 53   2             DO;
 54   3               Message_Ptr=@Break_MSG(0);
 55   3               Call Send_Msg;
 56   3             End;
 57   2           Else
                    Do;
 58   3               Char=Input(RxBuffReg) and StripTo7fh;
 59   3               Call MCO(Char);
 60   3             End;
 61   2           OutPut(RstIntReg)=Int_Reset;

 62   2         End Receive_Char;
```

231125-11

```
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*       Procedure:      Write character to 8256AH UART                    */
 63   1         Transmit_Char: Procedure Interrupt 45H;

 64   2           Status=(Input(StatReg) and SioRxRdy);
 65   2           If Status and SioRxRdy Then
 66   2             Char=(MCI And StripTo7FH);    /* strip to 7 bits                    */
 67   2             If Char >= 20H And Char <= 7fH Then   /* if char is ASCII output it */
 68   2               Output(TxBuffReg)=Char;
 69   2           OutPut(RstIntReg)=Int_Reset;

 70   2         End Transmit_Char;
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*      Procedure:      Write character to 856AH UART                      */
 71   1         Timer_2_4: Procedure Interrupt 46H;

 72   2           Message_Ptr=@CTS2_MSG(0);
 73   2           Call Send_Msg;
 74   2           OutPut(RstIntReg)=Int_Reset;

 75   2         End Timer_2_4;
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*       Message Output Procedure                                        */
 76   1         Send_Msg: Procedure;

 77   2           DCL     Message Based Message_Ptr (1) Byte;

 78   2           J=0;
 79   2           Do While Message(J) <> 0;
 80   3             Char=Message(J);
 81   3             Call MCO(Char);
 82   3             J=J+1;
 83   3           End;
 84   2           Return;

 85   2         End Send_Msg;
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */


                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /*      Main Program Body                                                */

 86   1         Call Init56;
 87   1         Do Forever;
 88   2         End;

                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

231125-12

231125-001

```
PL/M-86 COMPILER    MAINMOD

  89   1      End MainMod;


MODULE INFORMATION:

     CODE AREA SIZE     = 0235H    565D
     CONSTANT AREA SIZE = 00BEH    190D
     VARIABLE AREA SIZE = 0007H      7D
     MAXIMUM STACK SIZE = 0034H     52D
     280 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     31KB MEMORY AVAILABLE
     6KB MEMORY USED   (19%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```